ARI TECHNICAL REPORT TR-78-A9

Aspects of Effective Authoring Systems and Assistance: Recommendations for Research and Development.

by

Karl L. Zinn, University of Michigan

and

Alfred Bork, University of California at Irvine

Contracted by:

BATTELLE COLUMBUS LABORATORIES

Columbus, Ohio

SEPTEMBER 1978

Contract DAJC04-72-A-0001

Beatrice J. Farr, Project Scientist
Leon H. Nawrocki, Work Unit Leader
Educational Technology and Training Simulation Technical Area, ARI

Prepared for

U.S. ARMY RESEARCH INSTITUTE
for the BEHAVIORAL and SOCIAL SCIENCES
5001 Eisenhower Avenue
Alexandria, Virginia 22333

U. S. ARMY RESEARCH INSTITUTE
FOR THE BEHAVIORAL AND SOCIAL SCIENCES

A Field Operating Agency under the Jurisdiction of the Deputy Chief of Staff for Personnel

JOSEPH ZEIDNER
Technical Director

WILLIAM L. HAUSER
Colonel, US Army
Commander

Research accomplished
under contract to theDepartment of theArmy

Battelle Columbus Laboratories

NOTICES

FOREWORD

This work was performed as part of the Army Research Institute's (ARI) research program on the application of computer technology in education and training. The effort was initiated and funded during FY 75 within the Unit Training and Educational Technology Technical Area, under the direction of Dr. Frank J. Harris, Chief and Dr. Joseph S. Ward, Work Unit Leader. Responsibility for completing and documenting this work was assumed by the Educational Technology and Training Simulation Technical Area during FY 76.

The primary impetus for this undertaking was the nearly universal belief among members of the DOD research community that there was a need for more interaction between those engaged in research, those involved in creating software and developing authoring languages, and hardware vendors. One unfortunate consequence of this lack of communication was that system requirements for users (authors) were frequently overlooked to the extent of being detrimental to system effectiveness. In an attempt to ameliorate this situation, ARI convened a three day meeting so that selected representatives from each of the above mentioned domains could discuss developments as well as problems of mutual interest. The conference had multiple goals; it was directed toward facilitating information exchange and toward establishing suitable guidelines for applying computer technology to training needs, with military training as the focal point.

Through the Scientific Services Program of the US Army Research Office, a contract was let under Battelle Columbus Laboratories to secure the services of ten scientists and educators currently engaged in widely diverisfied CAI activities. These experts, as well as technical and user representatives from each of the services psychological research organizations or operational CAI activities were the primary conference participants. In addition, more than fifty individuals from the Department of Defense, other government agencies, private research groups and academia were invited to attend the first day of the meeting as observers. The conference was held 9-11 September in Alexandria, VA. During the first morning session representatives from the Army, Navy and Air Force gave formal presentations detailing both past and present programs relating to computer-based training. Considerable attention was also focused on current and anticipated problem areas. The afternoon was devoted to informal exchanges between the participants and observers. The remaining two days were spent in small-group problem-solving sessions which culminated in decisions regarding the topics of papers to be prepared by the participants subsequent to the meeting.

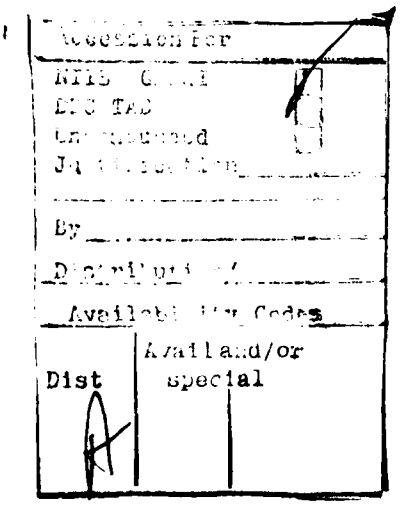
As initially envisioned, the working sessions were expected to focus almost exclusively on the authoring process. Although the major emphasis did remain as planned, during the course of the conference it

became clear that it would be more profitable to expand the scope beyond the original conception. In effect, the new agenda encompassed topics ranging from models which describe students, instructors and the learning process to sophisticated problems in artificial intelligence.

One of the primary goals of the conference was to attempt to have this diversified group of experts arrive at some consensus with respect to: defining user needs and requirements for authoring languages, identifying deficiencies within existing languages, delineating desirable characteristics for an ideal authoring language and establishing priorities for future research. Although somewhat less than consensus was reached, participants did identify a number of the most critical issues and offerred guidelines for research directed toward resolving the major problem areas.

Joseph Zeidner

JOSEPH ZEIDNER
Technical Director



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER TR-78-A9	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) ASPECTS OF EFFECTIVE AUTHORING SYSTEM AND ASSISTANCE RECOMMENDATIONS FOR RESEARCH AND DEVELOPMENT		5. TYPE OF REPORT & PERIOD COVERED 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Karl L. Zinn and Alfred Bork		8. CONTRACT OR GRANT NUMBER(s) DAJC04-72-A-0001 Task-Order-74-424)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Battelle Columbus Laboratories Columbus, Ohio		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 2Q763731A762
11. CONTROLLING OFFICE NAME AND ADDRESS Office of the Deputy Chief of Staff for Personnel, Washington, DC 20310		12. REPORT DATE September 1978 13. NUMBER OF PAGES 26
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Army Research Institute for theBehavioral and Social Sciences PERI-OK 5001 Eisenhower Ave., Alexandria, VA 22333		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Research monitored technically by Dr. Leon H. Nawrocki and Dr. Beatrice J. Farr Educational Technology & Simulation Technical Area, ARI.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer-Based Instruction
programming languages
Authoring
Instructional Systems

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This is one of a series of papers dealing with the authoring process and related problems in Computer-Based Instruction (CBI). It provides a brief historical background of computer based authoring systems, gives details of some of the contributions that the computer has made in the development of more effective teaching materials, outlines the most important aspects of the authoring process and provides some recommendations for future research and development.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE When Data Entered)

ASPECTS OF EFFECTIVE AUTHORING SYSTEMS AND ASSISTANCE: RECOMMENDATIONS FOR RESEARCH AND DEVELOPMENT

BRIEF

Requirement:

This paper is second in a series of reports emerging from a conference on Authoring Languages and Research Problems in Computer-Based Instructional Systems. The conference was sponsored and conducted by the Army Research Institute for Behavioral and Social Sciences as part of the Technology Base Work Program. It was included in the "DOD Integrated Plan for the Use of Computers in Education and Training."

Approach:

A three day conference was convened to determine the state-of-the-art and future research directions for authoring systems, particularly research issues relevant to improving the interface between computer-based instructional systems and instructional developers (authors). Participants consisted of ten technical consultants who were charged with identifying and reporting on major topic areas. Additional invited technical and user representatives (governmental, industrial and academic) participated either actively or as observers throughout the conference (the Appendix provides a list of participants).The first day was devoted to (1) formal presentations by military training system representatives describing current and planned computer-based instruction activities within the military, and (2) a roundtable discussion which delineated and defined major topic areas to be addressed. During the following two days, participants divided into four working groups. At the final session, each group presented a summary of their discussions relating to the key issues and approaches to authoring system research. Active participants selected topics for follow-on reports to be prepared subsequent to the conference.

Determinations:

Effective authoring is a functions of many factors, one of the most notable of which is the language in which the instructional material is programmed. Ease of entry is an important consideration. Early author languages did little more than adapt programmed instruction text for computer presentation. They were essentially frame-oriented in contrast to later languages which frequently contained specialized "logics" which permit separation of strategy from content. The most recent generative and mixed initiative approaches to CAI carry this separation even further.

A significant portion of future development in computer-based instruction is likely to be in the areas of generative systems and specially designed simulated environments for skill learning. This will

increase the demand not only for systematic pedagogy and instructional language development, but also for these special needs to be recognized in software design.

The areas needing particular attention can be derived by comparing the ideal authoring situation to current reality. Among these are: human factors consideration in designing appropriate keyboards and effective visual displays, possibilities for voice input in authoring systems, progress in natural language processing and increasing authoring capabilities via extensible languages.

Utilization of Findings:

Existing authoring languages for CAI fall short of meeting the needs of Army courseware developers, particularly in view of their relatively short training time and rapid turnover.

A research program to more closely integrate the requirements of subject matter experts and instructional programmers (authors) could profitably focus on better methods for: (1) program entry, (2) internal record keeping, and (3) program storage and modification. High priority should be assigned to devising appropriate sequences prior to using them with large groups of students.

CONTENTS

FIGURES

ASPECTS OF EFFECTIVE AUTHORING SYSTEMS AND ASSISTANCE: RECOMMENDATIONS FOR RESEARCH AND DEVELOPMENT

Programming languages for instructional use of computing should be considered within the larger context of systems by which materials are prepared and used. Many parts of the overall system affect the cost (both for development and for use) convenience, and effectiveness of development, and some shape the use of computer-based instructional materials as much or more than the formal language in which they were programmed. This report will consider, systematically, many aspects of effective authoring, in particular those for which research or further development may lead to more efficient computer-based education and training.

This report is arranged in four major sections: brief history; contributions to materials development; aspects of authoring systems; and recommendations for research.

BRIEF HISTORY OF COMPUTER-BASED AUTHORING SYSTEMS

Since the initial applications of computers in teaching, the designers of systems, programs and applications have given attention to authoring as something more than simply writing a computer program. For example, in the late fifties, early programs for teaching number systems, language vocabulary and analytic geometry were constructed in such a way that the teacher using the program could adjust the substance of the instructional program to obtain a variety of exercises and to serve the needs of different students. Some of these early designers clearly demonstrated the concept of a "driver" program which could be applied to a number of different information files in order to obtain, efficiently, a variety of substantive offerings for the student.

The first identifiable author language (called TIP: Translator of Interactive Programs) was written in the early sixties to ease the task of adapting a programmed instruction text in statistics for computer presentation. The programmer who was assigned the task of assembling information for the drum memory device of an IBM model 650 computer chose to develop a processor (a primitive compiler) which translated a programmed instruction text (including anticipated answers and feedback) into the structure required for the disk file. TIP provided a logic for computer delivery which was comfortable for authors coming from a background in programmed instruction.

The first version of IBM's COURSEWRITER language evolved from TIP. The approach to authoring encouraged by these early languages required arrangement of instructional material in frames; each frame included a question or other text, conditions by which a response from the student could be classified, and actions to be taken for each possible condition.

Concurrently, the PLATO Project at the University of Illinois was developing a somewhat different approach. Each key pressed by the student was treated separately so the student could be interrupted halfway through an answer based on what had been entered so far. Also, the author (lesson designer) was able to control the plotting of a point or the drawing of a line as a result of each individual key press. The next action taken could be based on a single key press, and the structure for accomplishing this (called a "mode switch") was the unifying concept. This approach is in contrast to that of a "frame" for COURSEWRITER. Since this language (called CATO) provided only the primitive elements, authors or their programming associates built within this language specialized "logics" for different courses or kinds of exercises. These logics represented a considerable advance along the same line of separation of strategy (in "drivers") from content (in information files) referred to above.

Later developments of instructional software for the PLATO computer-based education system went more in the direction of COURSEWRITER in that the author provided text marked as questions, anticipated answers, and specified actions to be taken. However, the PLATO design continued to leave open the nature of various actions that might be taken and the variety of displays which might be shown. For a long time, each new author might request some additional operations in the language specific to his purposes. Although the language (called TUTOR) has largely stabilized, it is still open ended. The library of operations is so large that few authors will learn all of them; but a consultant on the use of the language can advise new authors or authoring groups on the subset of operations which will be helpful to their purposes.

A quite different approach to authoring was taken at other universities. One of these is particularly well developed at the University of California at Irvine. Here, the author is invited to develop his own notation, usually a flow chart format combined with some convenient way to provide related text to be displayed. The actual programming is done by the technical support staff in whatever language is most convenient for the job at hand. At Irvine, most work is done in a macro language developed specifically for this purpose. At the University of Michigan, instructional uses are programmed in FORTRAN,. PL/1, APL, SNOBOL, BASIC, and only occasionally in a traditional coursewriting language such as Coursewriter or FOIL.

A more structured approach to individual author notations is seen in preprocessors designed for easier entry into whatever language is used. For example, COURSEWRITER is not convenient for many of its present uses. However, translators have been written to convert automatically into COURSEWRITER code other notations that authors find convenient.

Another version of easy entry carries on a dialogue with the author in order to be sure that all the information is provided in its proper place. The PLANIT System provided the first instance of this with on-line entry. A number of other systems have diverged considerably from the COURSEWRITER approach; in some, for example, the author may describe the curriculum material as a network of information rather than an essentially linear sequence of fit frames. The TICS system in use at MIT is a particular instance of this.

The TICCIT system developed by MITRE Corporation uses a software system defined by people at Brigham Young University. The intent was to build, into the software, specific instructional strategies found to be effective with the curriculum and level of student for which the system will be used. For a considerable period of time, the BYU Institute for Computer Uses in Education was particularly interested in providing the learner a certain kind of control. The student sets mastery goals and, in part, determines the amount of practice and occasions for testing to reach those specific goals. Because of a commitment to large-scale curriculum development and specific strategies, the TICCIT system adopted the separation of logic and content (something which had earlier been found to be efficient).

The generative approach to computer-assisted instruction carries the separation of logic and content still further. The designer of a system conceives strategies for: 1) identifying what the student needs and 2) delivering the information, exercises, quizzes or other material to him. Carbonell (1970) spoke of "mixed-initiative systems" in which the student could obtain answers to questions as well as check himself on substance. The information or substance of instruction is defined in procedures, and in an information network. In the initial versions of such systems the "author" needed to be very familiar with the procedures for representing information and objectives in the computer file. However, some progress is being made in providing for author interaction with the computer to build such data bases and procedures without requiring detailed knowledge of the techniques of artificial intelligence and information structures which are somewhat specialized. A significant part of future authoring is likely to be in this area of generative systems; designers of authoring and support procedures will have to attend to a number of special considerations.

Another area for development in the future is that of specially designed environments for learning a set of skills. For example, a teacher of management skills may design a simulated environment in which the student must develop and reliably execute those skills in order to perform well. Presently, the writing of management games or other simulations has no systematic pedagogy or instructional language. However, some of the special considerations in learning from this mode of computer use should be recognized in software design. Other factors relating to

the future needs and opportunities for authoring are taken up in the last section of this report.

CONTRIBUTIONS OF COMPUTING TO DEVELOPMENT OF MORE EFFECTIVE LEARNING MATERIALS

Inherent in one's decision to use the computer, is the advantage which is gained through a commitment to more careful attention to learning materials. The newness of computing, vis-a-vis other means of learning, combined with the apparent cost of computing for individual learning activities, tends to focus attention on the quality of materials. In addition to this secondary advantage, some aspects of the language, system and supporting services can make specific contributions to the quality of the learning materials under development. This section identifies some of those contributions which are closely associated with computer use.

Authoring systems such as the one designed for the TICCIT facility require the persons developing materials to identify, for the student, the objectives of instruction and various means for achieving them. The designers must consider what goals can be reached, and whether the materials and procedures offered to the student are sufficient. Futhermore, the computer system returns information to the developers organized in a way that clearly identifies both the success and shortcomings of the materials.

An authoring system (or programming language) which separates the instruction strategy from the learning materials can be used (within authoring system) in a way which contributes to the quality and perhaps the quality of materials produced. For example, it seems likely that the author or authoring team which selects specific tested strategies most suitable for each learning exercise, is going to be more productive and, on the average, more effective than one which spends lots of time working out each step in each instructional sequence. The entry of material according to an established format is also more convenient, thereby allowing the author to give more consideration to the suitability and effectiveness of the substance of instruction. Most important, perhaps, is the increased opportunity for revision of materials. The author or an aide can scan the data files for terms or phrases which need attention, can reset parameters in standard instructional procedures, or otherwise readily make adjustments.

Computer routines have been used to check the completeness of a teaching sequence. Branching, which is inherent in the logic, needs to be filled out by corresponding codes in the materials. A computer program can check to see that these exist. In general, the author may use the computer in drafting materials; for example, to keep track of branches in the instructional sequence which still need attention, to

alert one's self with reminders of directions for additional work previously noted, and otherwise to assist in drafting, testing and revision. This kind of assistance becomes increasingly important as more intelligence is incorporated within the computer procedures and complex data structures. Special kinds of computing assistance may be quite important in development of materials for generative programs and mixed initiative dislogues.

Computer assistance is potentially important in supporting authoring teams in which different functions are served by different individuals. For example, a complex presentation involving various media may require the assistance of technical experts as well as instructional specialists and subject matter experts. Computing files and procedures can be used to help coordinate these various efforts so that each contributor is able to put forth his best effort without being distracted by the need to attend to details of coordinating with others.

Some authoring systems make the collection of data rather easy; others almost preclude it. In nearly all approaches to computing in teaching, information about the instructional activity and achievements of the learners will contribute to improvement of the quality of teaching. The computer can be programmed and used in ways which help with this data collection for quality control and improvement.

When the computer is used as a medium for instruction, revising material should be easy. Although instructional materials for which the logic and substance are separated may, in general, be easier to correct and improve, any system can be helpful in the identification of errors and entry of corrections, changes and other improvements.

An authoring system can help or hinder the development of alternate versions of a teaching sequence. Good assistance from the system encourages authors to provide greater variety in the set of materials offered to meet the needs of different individuals. Provision for alternate versions is also helpful in research. Computer assistance facilitates preparation of materials for different treatments.

The list of contributions of computing to materials development could be extended and some of the points could be given in greater detail. However, the intent here is to indicate the range of opportunities--not to provide a complete analysis.

ASPECTS OF AUTHORING

A number of strategies have been developed for authoring computer-based instructional programs. This section describes some of the possibilities which have been used in practice, and introduces a few other theoretically possible mechanisms.

We are interpreting computer-based instruction in a very broad sense, including any use of prepared dialog programs that are used by students for learning purposes. The term "Authoring System" is used to indicate the entire process of preparation and revision of such programs. There are many variants possible, so the discussion must depart from logical order if it is to be reasonably complete.

For the purposes of classification, the following distinctions have been used:

A. What role does the author of the program play in the developmental process? What technical personnel assist? Does the author write computer programs? Does he or she prepare materials in some other way?

B. What notations or languages are used to write the program? Do the writers use charts, forms, data files, procedure descriptions or formal computer languages? Are the notations general-purpose or specialized for the instructional strategy or topic? In what representation (e.g. language, diagram or interaction) does the author review the materials?

C. How are programs entered? What preprocessors or translators are involved? What peripherals are used? Is the entry of needed elements prompted?

D. How is the program pretested before regular trials with students? Is some automatic checking of completeness and logic accomplished by machine? How is the programmer and/or author aided in reviewing the procedures and substance? What aspects of the authoring system facilitate preliminary testing with students?

E. How is the program stored and modified? What representation of the program is given to the author for review and revision? How is editing (revision) accomplished? To what extent can the computer version of the material serve as documentation? Can an author obtain advice while working on-line, whether automatically (from an "intelligent" manual for the authoring system), from a consultant working at another user terminal, or perhaps through a computer-based conference of authors working at different times (communicating via the "store-and-forward" function of computer-based communications).

F. What internal records are kept as the programs are used by students? What is the cost of saving and analyzing records? Are records convenient to use?

G. How is restarting handled for the student who returns after having been interrupted in the middle of a dialog?

H. What sort of built-in provision is made for gathering data (student responses) and rewriting a program?

The Author's Role.

With regard to the author's role in developing computer dialogs, there are several major considerations. These are listed in Figure 1. First, does the teacher himself write computer programs in any language at all, or does he/she engage in some other activity? Most projects assume that the eventual author will be writing programs for the computer, but a few systems do not. Several authors might be involved.

If the author of the dialog actually writes computer programs, two main distinctions are possible. First, the teacher can use a general purpose language that is already available, such as FORTRAN, BASIC, PL/1, APL, assembly language, or SNOBOL. Any powerful language with full capability is a possiblity. This general purpose language might be extended to include special facilities to ease the task of writing computer-student dialogs. Authors can also work within some specialized language developed expressly for the preparation of computer-based dialog materials; examples are: COURSEWRITER, PLANIT, and TUTOR. Preprocessors to translate materials into these specialized languages have also been used. Thus, the author might fill the forms which tell what question is asked, what answers to expect, etc. The designers of the languages developed just for authoring sometimes argued that no programming was involved--that the languages avoided most aspects of programming. Nevertheless, from the standpoint of the teacher (often a complete novice) the activity of using one of these languages would probably be viewed as just as much of a programming activity as writing in a language like FORTRAN--particularly as teaching programs become increasingly complex.

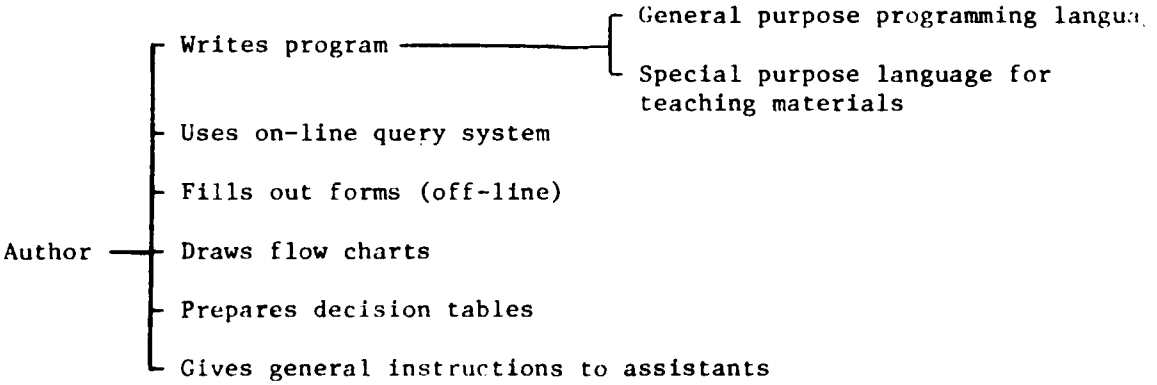


Figure 1. The Author's Role

It should be emphasized that the immediate question is not what language is used to get the program in the machine (which is, we believe, a separate issue discussed in the next section), but rather the question

of how the author of the program behaves in the authoring process.

Specialized languages for preparing dialogs can be broken into two classes. On the one hand (and this includes most of the actual material developed), we have languages which are languages in the conventional sense, needing to be written line after line in conventional program form. Most of the languages mentioned above, both general purpose and special purpose, are of this kind. However, another alternative, in which the author sits at a terminal and is queried by the system, is possible. PLANIT, SCHOLAR-TEACH, and DITRAN are examples of these querying modes. Clearly, some advance preparation is necessary for the author, so this approach tends to fit in somewhere between the case of using a programming language and the other techniques. A variant of query systems is the approach that has the author filling out forms, perhaps to be used in responding to queries directly, or through a secretary.

One alternative to writing programs is that the author or authors prepare a kind of flowchart, not a program in any particular language, but a graphic picture of what is to happen, and reflecting the pedagogical decisions that are made along the way showing how the program will respond to various student inputs. Figure 2 is an example of such a flowchart.

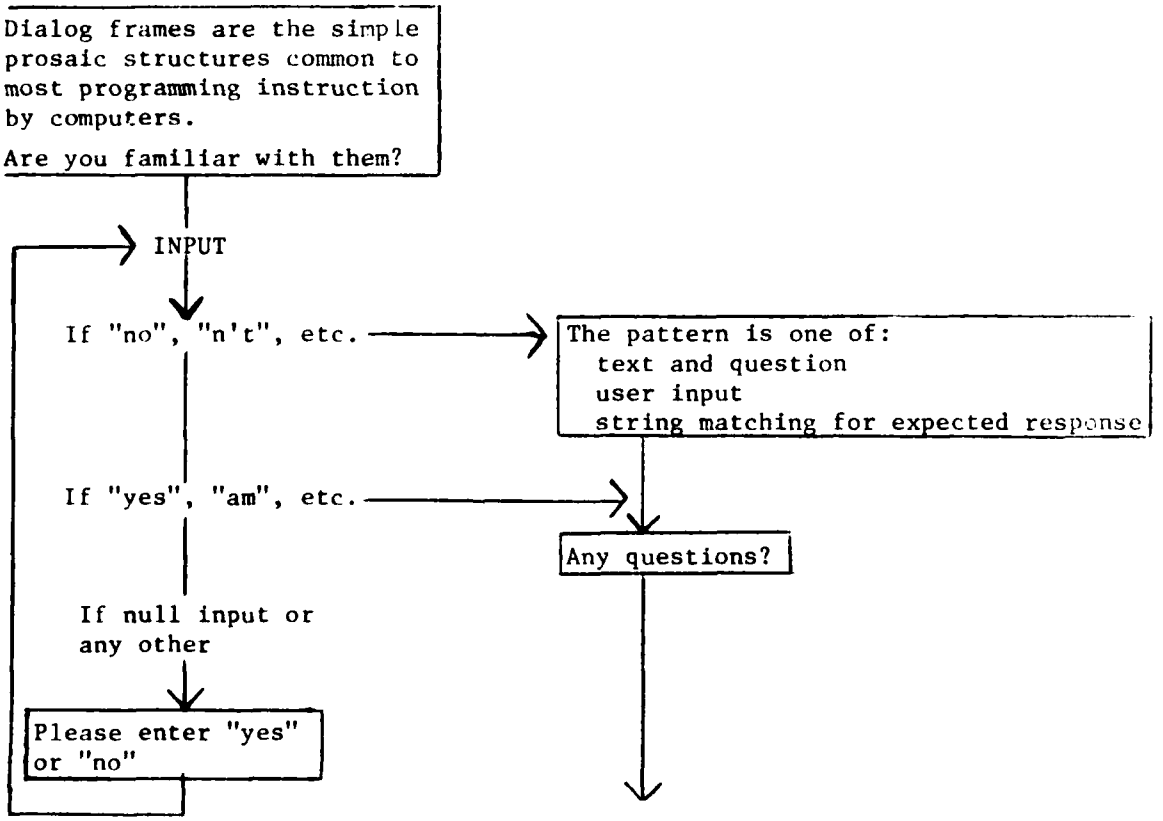


Figure 2. Sample Flowchart

Another possibility, similar to writing in a more or less loose flowchart format, is working with a decision-table format. Information conditions and outcomes are entered into a table from which the computer can determine the actions to be taken. Although this approach has been demonstrated, we do not know of any extensive amount of material being produced this way. Nevertheless, it remains a potentially feasible authoring style.

A further option is for the "primary" author to describe, with chalk and words, what he/she hopes to accomplish, leaving the many details to be developed by a competent assistant. With this approach the author does much less to define the details than when writing in the format of a fully developed flowchart. These details can then be confirmed and corrected by the primary author, who never uses any formal language other than that worked out between him/her and the assistant.

One area in which this "chalkboard" approach has been particularly useful is with "generative" material, based on problem generators or performance criteria programs; these programs often differ in pedagogical purpose from dialogs created in other ways. This kind of authoring activity is difficult to describe. Perhaps it is a special case of building a data base to fit a set of procedures. However, to date we have seen the "knowledgeable program" approach handled only by people who somehow combine subject expertise and computer science knowledge in one person. Indeed, these people are often computer science graduate students working on dissertations. (The most notable instance is the SCHOLAR system of Carbonell. It was his Ph.D. dissertation at M.I.T. that laid the cornerstone for some continuing work associated with the artificial intelligence and educational technology groups now at Bolt, Beranek and Newman). The author must be both the subject expert and system expert since the substance is structured within the data base according to techniques familiar only to computer scientists. Before Carbonell's untimely death, he was working on entry of material by people less knowledgeable about computer science.

Elliott Kaufmann at the University of Connecticut is also making progress on building systems which include some knowledge of the content and structure of the subject matter. The intent expressed by these various workers is to create a system which handles dialog with the student without the computer expressions and student input having been anticipated line by line.

Others working in the field include Michael Scott-Morton, MIT Sloan School, John Wexler, SUNY at Buffalo, Leonard Uhr, University of Wisconsin, David Levine, Rutgers, Adele Goldberg, PARC (Xerox), and Laurent Skilossy, University of Texas.

Language Used as the Source Language for Programs.

The range of possibilities has already been discussed in the previous section. Regardless of whether the author works in flowchart or decision-table form, either a general purpose language or a special teaching

language must be used as the language for the actual source programs out of which the student dialogs are created. These possibilities will be touched on briefly.

Several additional detailed distinctions between languages must be made. These are presented in Figure 3. First, a computer program, dialog or otherwise, need not be written in a single language. Several of the "specialized" languages for teaching are written in FORTRAN, and so they allow easy interfacing of FORTRAN and assembly subroutines. Assembly-based languages such as the Irvine dialog system also can be readily interfaced with FORTRAN or other compiler-based languages. The final program, then, can have parts originating in different source languages. If a general purpose, higher level language is used, the routine used repeatedly can be standardized as macros, subroutines or library code.

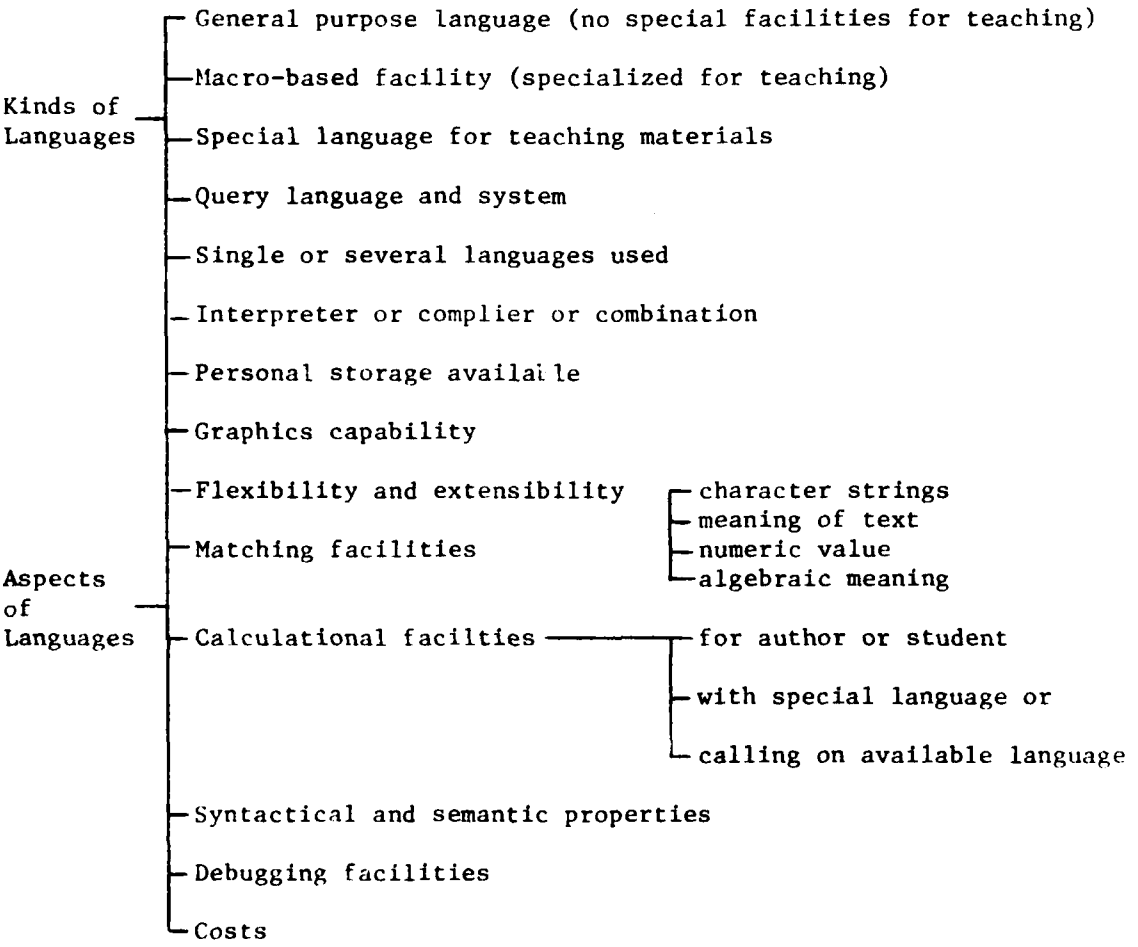


Figure 3. The (Source) Language Used

Another factor in language, assuming a non-querying system, is that of interpretive language versus compiled language. Interpretive language

have obvious advantages in instant feedback of error messages, and the ability to run the program immediately when entered. On the other hand, compiled languages are likely to produce more efficient run-time code, and thus, when the program is used by many students, save considerable computer time. Combinations are also possible, in which an interpretive implementation is used for the preparation stages, and a compiler for the final product, or where lines are compiled when first encountered and the code is stored.

When general purpose languages are used in developing teaching programs, the user has "unlimited" storage available which is unique to him; whatever limitations exist are functions of the core available, the operating system, the installation-imposed limits on individual users of the computer, and the disk storage available to users. Thus a computer-student dialog written in FORTRAN or assembly language might keep thousands of values (based on student input) computed at one point but held for use at some later stage of the program, or in another dialog. However, some of the languages devised directly for educational use have assumed severe limitations on personal storage, and so do not allow large amounts of personal information to be kept on hand. We can see that storage available to the individual user varies from one authoring system to another, a factor which can severely limit what is possible when using the computer as a teaching technique.

Other specialized and general purpose languages often assumed that the student was to receive alphanumeric output in some form. Recently, the development of inexpensive graphic terminals, which can draw lines at computer command, has opened powerful new teaching possibilities. Thus, graphic software is becoming an important component of contemporary authoring systems. Often "packages" exist for general purpose languages such as FORTRAN, at least for certain terminals; in some cases, this code is available from the terminal vendors, and in others from user libraries or developed locally. With the specialized teaching language facilities the graphic situation depends on the age and flexibility of the languages; older languages seldom considered graphics, and as long as it remains difficult to add to the facilities, that situation is likely to continue. But newer products (TUTOR, and the Irvine dialog facilities) have full facilities for drawing lines at graphic terminals, and some (usually limited) facilities for entering graphic data. The nature of what can be done, however, differs from case to case, and as such, needs to be individually examined when comparing languages and authoring systems.

The term "extensibility" often has a technical meaning when applied to programming languages. Here it is used in the general sense of adding facilities, of any type, to existing languages or sets of facilities. Early, specialized languages for developing computer-based teaching

material were based on a definite and often restricted view of how to use the computer in teaching. These languages did not lend themselves easily to extended uses not covered in the original plan. As a result, extensive internal calculation based on student input data was often difficult or impossible. More recent specialized languages, and many general purpose languages, allow anything that is possible with the computer; they do not assume a particular pedagogical strategy. However, complete freedom, particularly in a multipurpose environment, may not be desirable because the user may gobble up far too much of the system resources.

One way in which languages and systems can differ in very complicated ways is with regard to matching facilities (for recognizing student input). A number of distinct types of input must be dealt with. First, most dialog facilities allow key word or key string matching, perhaps with various logical constraints. Thus, a language may ask that the student input be one of several strings, followed by another string, but not containing a third string.

SNOBOL is an example of such a general, pattern-matching facility. Whether this matching is done at the whole word level, or whether matches on fragments of words are possible, is another source of variation. Some languages (TUTOR and PLANIT) have clever facilities which make allowances for spelling errors in such keyword searches.

A second type of technique for matching student input concerns natural language input, where an attempt is made to parse and "understand" the input, and respond to it. Although programs of this kind have existed for many years, their routine use in practical computer-based instruction involving large numbers of students is still in the future, if at all.

A third type of matching concerns student formulae, often containing derivatives, integrals, and various types of operators. Augmented matching can be sufficient in some situations. A simple algebraic formula, in some facilities, can be handled by substitution of numerical values, and comparison with correct values. This method, however, is insufficient when the formula is not algebraic in structure. A related task is that of using a student-input formula for internal calculations, again a parsing operation.

A fourth matching facility concerns the ability to deal with the student's numerical input, i.e., to find if it is within certain ranges. This input is often imbedded in text strings which must also be examined within the program; thus if "10 meters" is entered, both the numeric and alphanumeric information must be analyzed.

Matching is not simply a question of software, but has implications for hardware too. Some computer systems have powerful hardware instructions for rapid string matching, but these instructions are not necessarily

usable with a particular type of authoring software. Others must match byte by byte or word by word. Since hundreds of matches may take place each time a student enters an input (at least in some sections of the program) efficiency in these matters can be of considerable importance in determining how many students can be handled in the program.

Another difference between authoring systems concerns the question of numerical abilities. Here we should speak both of numerical abilities for the author, calculations he can request within the program which may be dependent on student input data, and numerical abilities given to the student when he is running a dialog. In both cases, these numerical calcualtions can be done either in a general purpose language, one commonly available on the machine for other purposes, or in a specially developed "language" just for the calculational purpose at hand. Some systems do not offer this capability, but many teaching systems have gone in the other direction, providing a new set of calculation facilities. Some systems have taken care of these needs by providing linkage for the author and/or the student to a higher level language already in widespread use, such as FORTRAN or APL. In such a case, the student can write long programs before returning to the dialog. It would also be convenient to be able to "pass" variables between dialogs and calculational programs; the APL "shared-variable" concept provides such a mechanism.

It is useful, in some situations, to relate the results calculated by the author's program to results calculated by the students. That is, the student may leave the dialog to do some work using a calculation language, and then return with certain results already identified for checking by the tutorial program. The advantage here is a reduction in the type of clerical and copying errors that follow from having students copy down results and type them again for the system.

Also, students may leave the dialog to write their own programs. If these are intended to arrive at a solution, and the result is incorrect, it is helpful if the dialog strategy has access to intermediate results and perhaps diagnostics coming from the programming and problem-solving efforts.

Surely, if one is using the dialog system to teach the student about programming, one wants to be able to get diagnostic codes back into the dialog in order that additional explanations will be given (beyond what is built into the language processor itself). One can provide procedural suggestions about how to solve the problem, write the program, or check it out.

A language for computer-based instruction can be judged on the same basis as other computer languages, so we can ask about some of the general syntactical and semantical properties. Newer languages like

ALGOL and PL/1 allow convenient block structure, an advantage in programming, debugging, and in reading programs obtained from other systems. Blocks of code, particularly within conditional statements, allow use of fewer GOTO commands; it has been argued recently that the GOTO is a common source of programming error. More generally, it is convenient to treat a very large program (and dialogs can be very large--several of the Irvine dialogs have load modules of over 100K words) as a group of subprograms (not necessarily subroutines in the normal sense). Another related useful syntactical feature is the ability to make variables local to a piece of the program, rather than having them be global in scope. Other aspects of data also need to be considered, such as the ability to dynamically allocate storage for data, and the ability to introduce new data types. Ability in start parallel processes may also be important. Other syntactical features need to be examined in comparing different authoring systems.

Different authoring systems provide considerably different debugging aids. They are often a function of the operating system in which the language is embedded. These aids range all the way from no assistance at all (beyond working through the initial code) to various tracing facilities of increasing sophistication. Thus a language like APL has built-in debugging facilities, allowing one to set traces and breakpoints in convenient ways, and allows the inspection of variables when errors occur. Systems that work with an efficient on-line assembly-language debugger such as DDT on the PDP10 or DELTA on the Sigma 7, provide extensive debugging facilities for the experienced programmer, but not necessarily ones which can be used by the novice. The question of who is doing the debugging, the teacher or a student programmer, is important in judging debugging facilities.

Method of Entering Program.

After a "program" exists in one of the forms suggested in the first section of this paper, it must be placed into the machine. Some of the variations are listed in Figure 4.

The first distinction between methods of getting programs into the computer initially is the classic one between timesharing systems and batch systems. Is the program to be punched into a series of cards and read into the card reader, or prepared off-line in some other fashion, perhaps with tape or disk units? Or is it to be entered interactively at the terminal? Some computers and languages allow both possibilities, but particular authoring groups will often favor one or the other. With a querying language, as discussed above, the information must be entered directly at the terminal. With an algorithmic language, either possibility may exist.

A second question relates to who is typing at the terminal, or using the keypunch, or other input device. Many such systems, particularly

interactive systems, have assumed that the teacher himself, the designer of the program, will also be the one who is sitting typing at the terminal, keypunch or other input device.

Other systems have assumed that programmers, perhaps students, will be doing this. Yet another variant is to assume the typing or keypunching will be done by secretaries, or by other personnel specially trained for this job.

A flexible approach, here and elsewhere, may allow any of the above variations and combinations. Thus the author may write out the program on paper, and give it to a keypunch operator or secretary to enter.

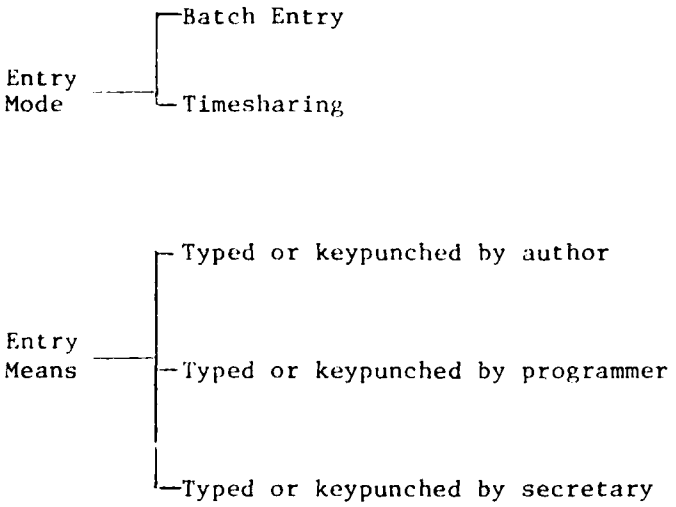


Figure 4. Means for Entering Program

Testing the Program Before Student Use.

Between the time that a working program (one that compiles or assembles without error) exists on the machine, and the time it is used with large groups of students, some testing will generally proceed. This is perhaps the part of authoring systems that is the least formalized, and has had the least energy devoted to it. Many small modifications are often required at this stage to produce even a modestly interactive program, so the issue is important. Some of the possibilities for pre-testing are listed in Figure 5.

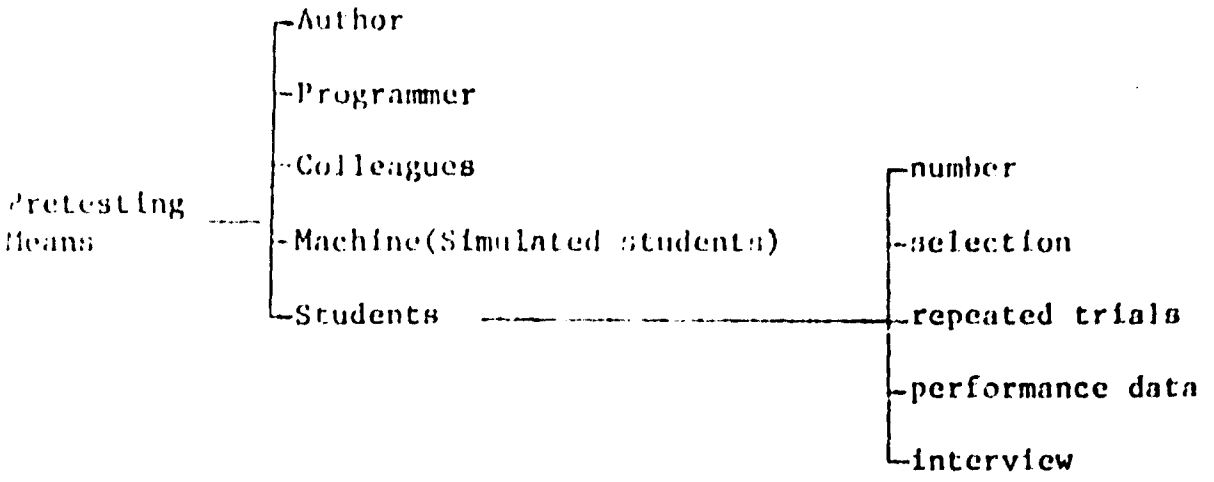


Figure 5. Testing Before Use

Initial testing is likely to be performed by the teacher and associated programmers, although this is not essential. If any further testing goes on at all at this level--and in many places it does not--it often consists of simply gathering a few students (perhaps out of the halls) and running them through the material, observing what happens either directly, or using built-in storage facilities (to be examined in the Internal Record Keeping section). Often, colleagues are used in this testing, in addition to students who may or may not be representative of the target audience. Small or large groups of students may be used; thus the author may work directly with two students, obtaining quick feedback, or he may have large groups pre-test the dialog. Some more formal testing procedures may also be employed. It is possible to use the program in the normal fashion, as it will be used ultimately, except with a smaller group of students. Then it can be rewritten once before being used with larger groups. Automatic methods of running through the program, with random filed inputs generated by the computer itself, have also been suggested; it is not clear whether these have actually been used in practice, and they would not provide a complete test, in any case.

A description of approaches should include some mention of timing, number of students, kind of data, and particularly the role of the author. One approach is for the author to do very little before calling in at least two students to try it. In most instances, the author works in the same room, observing the students at work, and is available to answer their questions and to sit down and discuss it with them at the end of the session. The author then rewrites that section and goes on to the next, hopefully bolstered by his experience with the first two students. This approach, of having a very close cycle of test-revision with many batches of students in small numbers, is costly in author time but contributes to an empirical base for the validity of the curriculum materials. The data are quite casual and dependent upon the ability of students to express themselves and the skill of the author in identifying their difficulties.

Another approach uses larger numbers of students and is more dependent upon a clear description of objectives and procedures to get one through. After a large amount of material has been produced, it is tried by twenty or more students. The results are likely to be taken in summary form, saving the author the time of interviewing each student. The saving in author time, however, costs the project something, namely less richness of data.

Summary information derived by the computer (or technicians) should identify weak points in the instructional procedure; the author is looking for per cent wrong, particular wrong answers, and peculiar connections or paths. Presumably if the data are unclear, the author can turn to individual students for explanation.

The role of the author in all this should be spelled out. He may be very much involved himself, doing tutoring and extracting information from those experiences; he may be quite dependent on technical assistance from the machine (or human assistant) to collect and summarize data for him.

Program Storage and Modification.

The source program is not a completely finished document at any time; it is always subject to revision. In the case of an interpretive language facility, the source program will be the only one available, but in other cases, the program can exist in one of several forms. It can be stored outside the machine in the form of cards or magnetic tape. Internal storage is likely to be disk storage. When the program is actively being modified, it is convenient to keep it available in disk storage, although this is not possible in all systems. An authoring system in which the author must spend half a day locating the card deck, getting it back in the machine, etc., is clearly not as convenient as one in which the author can simply specify, at a user terminal, the program to be edited.

All the educational programs must be extensively edited during the development period. Editing systems are an important component of any authoring system. They often exist on the computer, independent of the activities of producing educational materials. Such systems for editing exhibit a wide variety of flexibility and aptness, with many variables. Although they are often not considered part of the teaching software, they are, nevertheless, an extremely important part of most authoring systems, and can greatly affect the convenience of use of these systems.

We might distinguish a number of features of editing systems, although no full discussion can be presented. First, is it possible to replace lines? (This might be called "bare minimum.") Is it possible to delete lines? Is it possible to conveniently change a few characters in a line, or in a large group of lines? Is it possible to reorder the

code, as is often necessary in a computer program? Is it possible to merge parts, or all, of different files, or to break up a single file into a collection of files? Does the editing system protect the casual user, or does it make it all too easy to wipe out sizeable portions of the program with a typing error in one place? Can the editing system be used by a skilled secretary with minimal training in its use, or does it require a computer professional?

Another important issue is documentation. The programs created are likely to be large and complex. They should be frequently modified to become more interactive and to deal with errors. Often, this modification will be done by people who did not write the original program, so full careful documentation is essential. Some of this can be supplied by the author, while other aspects may be part of the authoring system.

One approach is for the author to prepare documentation as he develops the program. In the case of flowcharts, this is the essence of the author's notation and likely to be carried along unless or until it becomes more convenient for him to work with the real representation within the machine. Nevertheless, when the program is finished, it may be worthwhile to bring the flowchart up to date to serve as documentation to share with other users and reviewers.

If the author stays distant from the representation of the program within the machine, perhaps the documentation would be generated in part by computer (as in a listing of problem headings and branches and such) or by a technical assistant working from the program listing itself.

Internal Record Keeping.

The saving of student responses is considered under Feedback and Revision, but other record keeping can go along with running dialogs. First, a complete authoring system should keep records of dialog usage--who runs which dialogs, and for how long. We may also want to record full error information when a program malfunctions. Long dialogs are complex programs and so may still have unknown problems that will only appear occasionally; the details of the error (the type of error, where it occurred) can be useful in finding the problem.

Another type of record keeping can affect the performance of the dialog. It may be desirable to access and process information from files, perhaps entered in running the program, perhaps not. Thus, an educationally oriented retrieval system, such as the Dartmouth IMPRESS system, can allow student access to very large structured data bases.

A third type of record keeping concerns communications between dialogs, either one user with several dialogs or one dialog with several users. In the early days of using computers in education, each dialog was a world in itself; a dialog might refer to another, but it would not

have access to information about student performance in other dialogs. It might, for example, be useful to know if a student currently using a dialog has passed some particular point in a previous dialog, or even how he/she responded to a certain question in another dialog. A complete authoring system can make provision for such passing of information from one program to another.

Passing information from one user to another is useful in multi-person games played on a computer. PLATO does this, occasionally, so that one user will know how successful preceeding ones have been. These data are part of "program storage" (in the Educom report specifications) since the data are really part of the information about the program and its uses, rather than information about individual students.

Restart Capability and Other Student Options.

For various reasons, a student may not complete a dialog in one sitting, particularly if the dialog is long and complicated. An authoring system may provide restart facilities so that the student does not necessarily have to start from the beginning the next time around. Either the student or the author may terminate the session.

A number of strategies exist; they are listed in Figure 6.

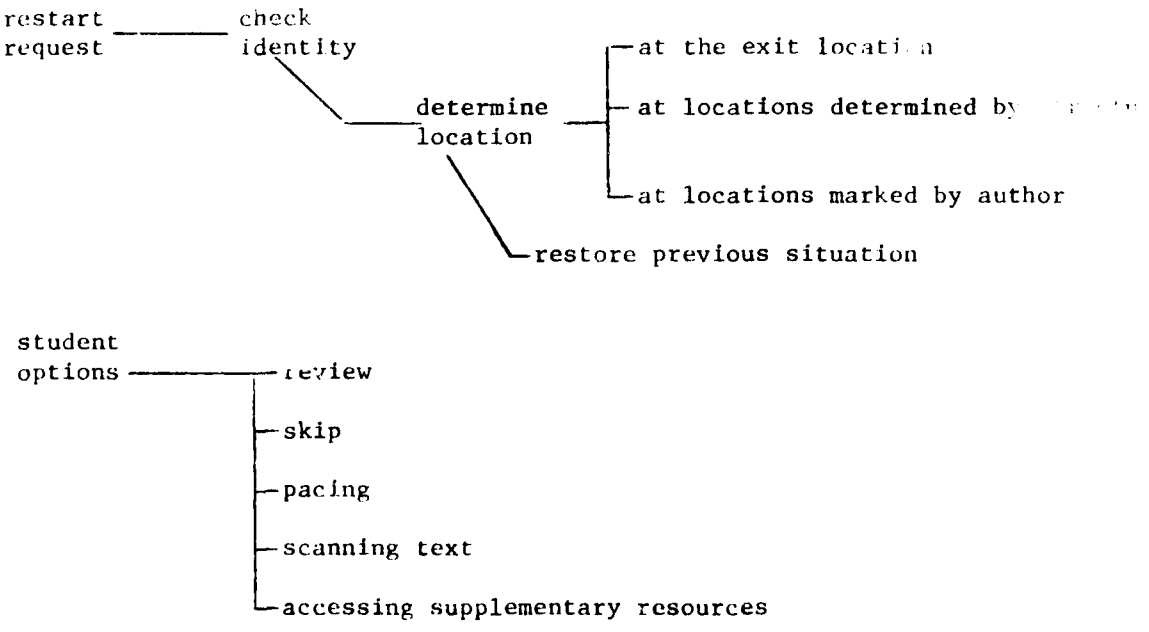


Figure 6. Restarting an Interrupted Session

The first decision is who is to start again. The program can access the account number of the student, or it can ask for a student-entered identification. If the student is suspected to be a repeat, it is still useful to ask if this is the case; students will, for example, often enter a personal identification such as BILL, and several students may use one account.

The next decision is where to restart. At least three possibilities can be envisioned, assuming restart. The student can be restarted precisely at the previous program exit, usually at the point where the computer is waiting for input; the problem is that the student may not remember the context. The second case assumes that restart points can be determined by the structure of the program; if it is composed of a number of subprograms the student can restart at the beginning of the sequence last used. The final situation, most flexible, is to allow the teacher preparing the dialog to specify just where restarts are possible; the student will begin again at the last such point finished.

In any case, a restart facility, if it is to be highly useful, should completely restore the previous status. All the internal variables should be set as before, and the same parts of the program should be in core. Implementation depends on file facilities.

Restart capability is only one student option. Others are also desirable in some situations. The student may wish to review certain material, may want to skip ahead, may want additional examples, or may want to control over pacing (the rate at which material is presented)

The notion of restart should be expanded to cover all manner of student options as well as the way in which the author sets limits on these options. For example, one would include provisions for the student: to skip ahead, go back and review, move to supplementary resources such as calculation or glossary, and scan or search about through the text for keywords or other indicators of substance. The accompanying mechanisms of concern to the author are a computer-based context (which may be present or determined at execution time), and a provision for selecting our parts of the text which either are identified by tags placed by the author, or selected by some other conditions such as depth within the logical structure, nature of the question format, etc.

Feedback and Revision

A computer dialog, as it is initially written, is seldom highly interactive. It will require large amounts of student use, plus several rewritings, before it will be highly responsive to many students in a completely free fashion. Of course, if the dialog restricts input heavily, this problem does not arise. But in many situations, a free-form dialog must be heavily used and revised before it is effective.

Some considerations in feedback and revision are listed in Figure 7.

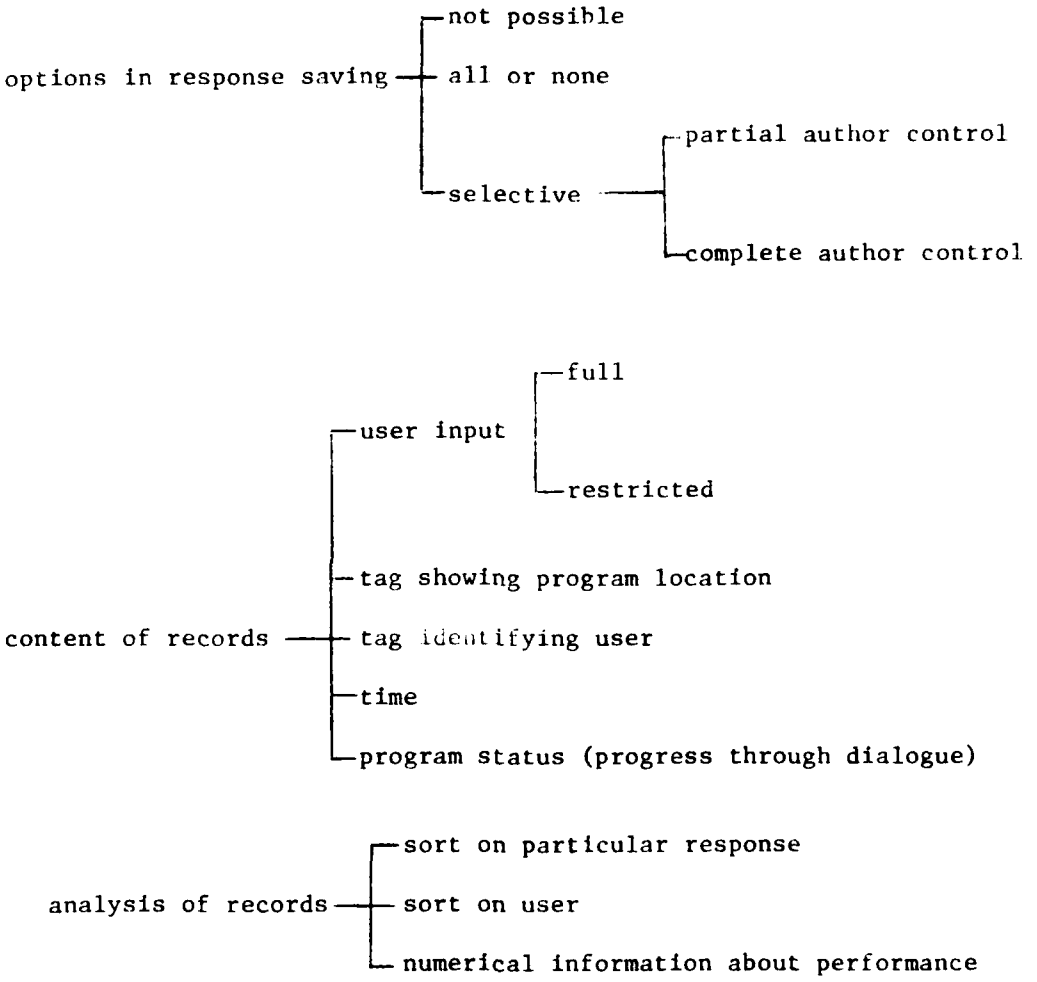


Figure 7. Feedback and Revision

The first major distinction relating to feedback in authoring systems is whether the software permits saving student responses in disk (or possibly tape) files. Clearly, the computer (with suitable hardware) has the ability to save student responses, but not all authoring systems allow this to happen. Furthermore, among those that do, several variants can be found. One possibility is an "all or nothing" situation--that is, all responses can be saved or, perhaps, none. On the other hand, the circumstance in which what is saved may be selective, fully under the author's control. In this last case, the author might specify that certain trivial responses not be saved at all, while in other situations the author may choose to save the student responses that the program could not analyze.

To attempt to save all responses, particularly in programs used with large groups of students, produces enormous files, perhaps too large for the machine. Just what is saved, is also a feature of authoring systems. With regard to student input, we inquire as to whether a full input, perhaps in places many lines long, can be stored, or if, as with some systems, there are limitations. The stored records also may contain various other "tags." One essential thing is to identify the place in the program where storage occurs, since student inputs may occur in hundreds of places in the code. It is also useful, on occasion, to identify the input as to student, so that the author can follow the progress of a particular student through the dialog. In a similar vein, the exact time of each student input may help in understanding what a student is doing. If one file is used for several dialogs (impractical for wide usage) the dialog must also be identified.

Another helpful operation involves the student path--or progress through the dialog: what branches did he/she take, what program facilities were used, how many times were retries necessary at a certain location?

Information of the type just suggested is more usable for the author who is rewriting material, if auxiliary programs exist to aid in massaging the raw data into other forms. Sorting is one such function. Student responses need to be sorted both with respect to the program location and with respect to users. Program progress also can be sorted, and perhaps the data can also be used to obtain statistical information about how the dialog is used.

Authors can be overwhelmed by data. Representation is critical, and raw data typically are not used at all. The nature of data reduction has proved to be a very expensive process, always in need of revision, and even still, not much used by authors.

The saving and manipulating of data reflecting student use is only one aspect of the revision problem, the one to which the computer contributes. Provision still must be made, in current authoring systems, for the teacher to use these data in reworking the program. Authoring systems in current use differ widely with respect to how much in the way of human resources they allocate to revision. As with initial testing, it is hard to formalize the requirements for revision.

RECOMMENDATIONS FOR RESEARCH

This final section lists areas and aspects needing particular attention in research programs and specific development activities. The list is derived from a comparison between the ideal situation (aspects of authoring and desirable contributions of computing) and the reality of current systems as seen in technical reports, manuals and publications.

Engineering

Human factors considerations in interactive computing need considerable attention before the responsiveness of the computer can be appreciated and put to full use. Today, most successful authors' computer-based learning materials build on long experience with computers and exhibit considerable tolerance for the shortcomings of the software and user terminals. What makes a keyboard effective for different kinds of users? What are the parameters of a usable display? In what ways can the load which interactive computing places on the memory of the user be alleviated?

What new kinds of visual displays may be created to extend the effectiveness of the author? For example, how can data and procedures be displayed to the materials designer in order to show complex structures in full detail one time, and in various simplifications at other times?

What is the proper role for speech input on authoring systems? In what learning environments will speech input be most helpful? What new dimensions of input processing will become available to the author, and how will classification of input be specified?

How will communication within a group of authors and reviewers be facilitated?

Information Science

New knowledge about information structures may contribute important tools for users of authoring systems, but many questions remain: What structures contribute to more efficient storage, more rapid retrieval, and more convenient access by the user? What structures open up new possibilities for representation of information, growing networks of knowledge, and opportunities for students and other users to add to the data base?

Extensible language hold some promise for increasing authoring capabilities in significant ways. In what ways can language be made to adapt to the changing needs of authors, working individually or in groups? How can a balance be achieved between: 1) adapting to the needs and requirements of individual subject areas or authoring groups, and 2) establishing general conventions, notations and procedures upon which all users of a family of languages can build?

Natural language processing offers considerable opportunity for extending access to automatic information processing. What would a system look like if the designer of materials needs only to speak or handwrite directions for setting up learning sequences? What additional problems need to be solved, e.g. speech input, disambiguation, and many instructional design considerations? What are the implications of

natural language processing for interaction with learner? For example, how does the role of the materials designer change when the system "understands" the student?

Instruction Science

Surprisingly, the area which has made the least impact to date on the use of computing in teaching is that of instruction science. The potential is there, but the methodology needs refinement. Eventually, the contribution will be felt. In the meantime, we still need to know what instructional strategies can be prescribed for particular learning tasks with some confidence of achieving near to optimum results with the objectives and students for which the materials were prepared? What kinds of learners most benefit from various kinds of computer assistance? In what ways can a computer program match instructional materials with learning styles?

Pedagogy (Disciplines)

The largest advances in the near term are likely to come in the area of pedagogy specific to the disciplines being taught. For example, new tools of information processing will undoubtedly open up new opportunities for effective learning activities. Here, too, though, many questions remain: Will information structures newly developed for research and scholarly work in a discipline be carried over to corresponding learning activities? What procedures of analysis (in an area of study) correspond to learning goals for understanding process and structure in that area? The right tools can give an advantage of two or three or even ten times the "learning" without computer assistance.

APPENDIX

PARTICIPANTS

Mr. Avron Barr
Institute for Mathematical Studies in the Social Sciences
Stanford University, Ventura Hall
Palo Alto, CA 94305

Dr. Alfred Bork
Department of Physics
University of California
Irvine, CA 92664

Dr. John Brackett
SofTech
460 Totten Pond Road
Waltham, MA 02154

Dr. Victor C. Bunderson
Institute for Computer Uses in Education
Brigham Young University
Provo, UT 84601

Mr. Frank Dare
CAI Project
USA Ordnance School and Center
Aberdeen, MD 21005

Mr. Wallace Feurzeig
Bolt, Beranek and Newman
50 Mouton Street
Cambridge, MA 02138

Dr. Dexter Fletcher
Navey Personnel Research and Development Center
San Diego, CA 92152

Mr. Ed Gardner
Air Foce Human Resources Laboratory
Lowry AFB, CO 80230

Dr. Roy Kaplow
Massachusetts Institute of Technology
Room 13-[illegible]0[illegible]
Cambridge, MA 02139

Mr. Don Kimberlin
Office of Project Manager
Computerized Training System
Ft Monmouth, NJ

Mr. George Lahey
Navy Personnel Research and Development Center
San Diego, CA 92152

Mr. Hal Peters
Hewlett-Packard
11000 Wolf Road
Cupertino, CA 95014

Dr. Mortenza A. Rahini
Department of Computer Sciences
Michigan State University
East Lansing, MI 48823

Dr. Martin Rockway
Air Force Human Resources Laboratory
Lowry AFB, CO 80230

Dr. Robert Seidel
HumRRO
300 N. Washington Street
Alexandria, VA 22314

Mr. Robert H. Simonsen
System Development Technology
Boeing Computer Service
Seattle, WA 98108

Dr. Lawrence Stolurow
Division of Educational Research
State University of New York
Stony Brook, NY 11790

Dr. Paul Tenczar
Computer-Based Educational Research Laboratory
Urbana, IL 61801

Dr. Karl Zinn
Center for Research in Learning and Teaching
University of Michigan
109 East Madison Street
Ann Arbor, MI 48104